Complexity of Multi-Value Byzantine Agreement [*]

Guanfeng Liang and Nitin Vaidya

Department of Electrical and Computer Engineering, and
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
gliang2@illinois.edu, nhv@illinois.edu

Technical Report

June 11, 2010

Report Documentation Page		*Form Approved* *OMB No. 0704-0188*

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **11 JUN 2010**	2. REPORT TYPE	3. DATES COVERED **00-00-2010 to 00-00-2010**
4. TITLE AND SUBTITLE **Complexity of Multi-Value Byzantine Agreement**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Illinois at Urbana-Champaign,Department of Electrical and Computer Engineering,Coordinated Science Laboratory,Urbana,IL,61801**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT **Same as Report (SAR)**	18. NUMBER OF PAGES **12**	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**			

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

1 Introduction

In this paper, we consider the problem of maximizing the throughput of Byzantine agreement, given that the **sum capacity** of all links in between nodes in the system is *finite*. Byzantine agreement (BA) is a classical problem in distributed computing, with initial solutions presented in the seminal work of Pease, Shostak and Lamport [27, 20]. Many variations on the Byzantine *agreement* problem have been introduced in the past, with some of the variations also called *consensus*. We will use the following definition of Byzantine agreement (Byzantine general problem): Consider a network with one node designated as the *sender* or *source* (S), and the other nodes designated as the *peers*. The goal of Byzantine agreement is for all the fault-free nodes to "agree on" the value being sent by the sender, despite the possibility that some of the nodes may be faulty. In particular, the following conditions must be satisfied:

- **Agreement:** All fault-free peers must agree on an identical value.

- **Validity:** If the sender is fault-free, then the agreed value must be identical to the sender's value.

- **Termination:** Agreement between fault-free peers is eventually achieved.

Our goal in this work is to design algorithms that can achieve optimal *throughput* of agreement. When defining throughput, the "value" referred in the above definition of agreement is viewed as an infinite sequence of *information* bits. We assume that the information bits have already been compressed, such that for any subsequence of length $n > 0$, the 2^n possible sequences are sent by the sender with equal probability. Thus, no set of information bits sent by the sender contains useful information about other bits. This assumption comes from the observation about "typical sequences" in Shannon's work [30].

At each peer, we view the agreed information as being represented in an array of infinite length. Initially, none of the bits in this array at a peer have been agreed upon. As time progresses, the array is filled in with agreed bits. In principle, the array may not necessarily be filled sequentially. For instance, a peer may agree on bit number 3 before it is able to agree on bit number 2. Once a peer agrees on any bit, that agreed bit cannot be changed.

We assume that an agreement algorithm begins execution at time 0. The system is assumed to be synchronous. In a given execution of an agreement algorithm, suppose that by time t all the fault-free peers have agreed upon bits 0 through $b(t) - 1$, and at least one fault-free peer has not yet agreed on bit number $b(t)$. Then, the agreement *throughput* is defined as[1] $\lim_{t \to \infty} \frac{b(t)}{t}$.

> **Capacity** of agreement in a given network, for a given sender and a given set of peers, is defined as the supremum of all achievable agreement throughputs.

2 Related Work

Prior work on agreement or consensus: There has been significant research on agreement in presence of *Byzantine* or *crash* failures, theory (e.g., [20, 24, 1]) and practice (e.g., [5, 4]) both. Perhaps closest to our context is the work on *continuous consensus* [26, 10, 25] and *multi-Paxos* [19, 5] that considers agreement on a long sequence of values. For our analysis of throughput as well, we will consider such a long sequence of values. In fact, [5] presents measurements of *throughput* achieved in a system that uses multi-Paxos. However, to the best of our knowledge, the past work on multi-Paxos and continuous

---

[1]As a technicality, we assume that the limit exists. The definitions here can be modified suitably to take into account the possibility that the limit may not exist.

consensus has not addressed the problem of optimizing throughput of agreement while considering the *capacities of the network links.*

In [8], it is proven that, without authentication, $O(n^2)$ bits are necessary to be communicated in total, in order to be able to agree on just 1 bit in the worst case when $t < n/3$. Algorithms have been derived to achieve this quadratic lower bound [2, 6]. Efforts have also been dedicated into tolerating more than failures and reducing the quadratic lower bound from [8], with the help of a public-key infrastructure and/or allowing a small probability of error [28, 9]. However, all these algorithms have bit complexity $\Omega(n^2)$. In [11], a multi-valued BA algorithm is introduced with complexity $O(nl+n^3(n+\kappa))$, where κ denotes a security parameter The quadratic lower bound is overcome by reducing a BA with a long message of l bits to BA with much shorter messages, using a universal hash function and allowing a small probability of error. On contrary, our algorithm is able to achieve agreement with a lower complexity even in the worst case.

Past work has explored the use of error-correcting codes for asynchronous consensus (e.g., [13]). Our algorithms also use error detecting codes, but somewhat differently.

Prior work on multicast using network coding: While the early work on fault tolerance typically relied on replication [7]or source coding [29] as mechanisms for tolerating packet tampering, *network coding* has been recently used with significant success as a mechanism for tolerating attacks or failures. In traditional routing algorithms, a node serving as a router, simply forwards packets on their way to a destination. With network coding, a node may "mix" (or *code*) packets from different neighbors [21], and forward the coded packets. This approach has been demonstrated to improve throughput, being of particular benefit in *multicast* scenarios [21, 17, 12]. The problem of *multicast* is related to *agreement*. There has been much research on multicast with network coding in presence of a Byzantine attacker (e.g., [3, 14, 15, 32, 16, 18]). The significant difference between Byzantine agreement and multicasting is that the multicast problem formulation assumes that the source of the data is always fault-free.

In our previous technical reports [31, 23, 22], we characterized the agreement capacity of general four-node networks with finite link capacity (some links may not exist). Byzantine agreement algorithm that achieves throughput arbitrarily close to the capacity are proposed for both complete and incomplete four-node networks.

3   Models

We assume a synchronous network fully connected network of n nodes, the IDs (identifiers) are common knowledge. We assume that all communication channels/links are private and that whenever a node sends a message directly to another, the identity of the sender is known to the recipient, but we otherwise make no cryptographic assumptions. We assume a strong adversary. That is, the adversary has complete knowledge on the BA algorithm and the information being sent by every node. The adversary can take over nodes at any point during the algorithm up to the point of taking over up to a $t < n/3$ nodes, including the source. The compromised nodes can engage in any kind of deviations from the algorithm, including false messages and collusion, or crash failures, while the remaining nodes are good and follow the algorithm.

4   Results

All our results are about achieving agreement deterministically, which means that under our algorithm, it is impossible for the good nodes to decide on different values.

- We show that there exists an algorithm which computes Byzantine agreement on an l-bit message in a network with n nodes and at most t faulty nodes, and uses $\frac{n(n-1)}{n-t}l + l^{1/2}O(n^4)$ bits of communication.

If we let l approach infinity and consider the average number of bits used for agreeing on 1 bit, then we have the following result on the sum capacity

- For the network with n nodes and at most t faulty nodes to achieve agreement throughput of R bits/unit time, it is sufficient to have sum capacity $> \frac{n(n-1)}{n-t}R$.

5 Byzantine Agreement Algorithm to Tolerate up to t Faults

The proposed Byzantine agreement algorithm progresses in generations. In each generation, $(n-t)c$ bits are being agreed upon. Normally (when no failure is detected), the length of each generation is chosen such that slightly more than $n(n-1)c$ bits can be transmitted in the network. The algorithm starts assuming no node is faulty.

5.1 Operations when no failure detected

Round 1: The source node 0 divides the $(n-t)c$ information bits into $n-t$ packets of size c bits, each packet being a symbol from $\mathrm{GF}(2^c)$. Then node 0 encodes the $n-t$ packets of data into $2(n-1)$ packets, each of which is obtained as a linear combination of the $n-t$ packets of data. Let us denote the $n-t$ data packets as the data vector

$$\tilde{x} = [x_1, x_2, \ldots, x_{n-t}] \tag{1}$$

and the $2(n-1)$ coded packets as

$$y_1, y_2, \ldots, y_{2(n-1)}. \tag{2}$$

For the correctness of the algorithm, these $2(n-1)$ symbols (or packets) need to be computed such that any subset of $n-t$ encoded symbols constitutes independent linear combinations of the $n-t$ data packets. As we know from the design of Reed-Solomon codes, if c is chosen large enough, this linear independence requirement can be satisfied. The weights or coefficients used to compute the linear combinations is part of the algorithm specification, and is assumed to be correctly known to all nodes a priori. Due to the above independence property, any $n-t$ of the $2(n-1)$ symbols - if they are not tampered - can be used to (uniquely) solve for the $n-t$ data symbols.

In round 1, node 0 transmits 2 packets y_i, y_{n-1+i} to node i. Other nodes do not transmit.

Round 2: Each peer node i sends packet y_i to every other peer node.

Round 3: Each fault-free peer finds the solution for each subset of $n-t$ packets from among the packets received from the other nodes in rounds 1 and 2. If the solutions to the various subsets are not unique, then the fault-free peer has detected faulty behavior by some node in the network. In round 3, each peer broadcasts to the remaining $n-1$ nodes a 1-bit notification in indicating whether it has detected a failure or not – to agreement among all n nodes on these 1-bit indicators is achieved by using an efficient traditional Byzantine agreement algorithm (e.g. the algorithm [6, 2]). Since less than $1/3$ of the nodes can be faulty, using this traditional algorithm, all fault-free nodes obtain identical 1-bit notifications from all the peers. If none of the notifications indicate a detected failure, then each fault-free peer agrees on the unique solution obtained above, and the execution of the current generation of the algorithm is completed. However, if failure detection is indicated by any peer, then an "extended round 3" is added to the execution, as elaborated soon.

Theorem 1 *Misbehavior by a faulty node will either be detected by at least one fault-free peer, or all the fault-free peers will reach agreement correctly.*

Proof: Node 0 is said to misbehave only if it sends packets to the peers that are inconsistent – that is, all of them are not appropriate linear combinations of an identical data vector $\tilde{x}$. A peer node i is said to misbehave if it forwards tampered (or incorrect) packets other than y_i to the other peers.

Source is faulty: Since the source node 0 is faulty, there are at least $n - t$ fault-free peers. From the way the peers forward their packets, it is easy to see that among the packets the fault-free peers have received, at least $n - t$ are identical. Thus, the fault-free peers will not agree on different data symbols (since the agreed data must satisfy linear equations corresponding to all received packets), even though node 0 and up to $t - 1$ peers may be misbehaving.

Source is fault-free: When source is fault-free, there are at least $n - t - 1$ fault-free peers. Similar to the previous case, among the packets a fault-free peer have received, at least $n - t - 2$ are received from fault-free the other peers and must be correct. In addition, every peer has also received two correct packets from source node 0. Together, every fault-free peers has receives at least $n - t$ correct packets, which has a unique solution $\tilde{x}$. Thus, the fault-free peers will not agree on any data vector other than $\tilde{x}$, even though up to t peers may be misbehaving. □

Extended Round 3: When failure detection is indicated by any peer in round 3, an extended round is added subsequent to a failure detection. We also refer to this extended round as the "broadcast phase". During the broadcast phase, every node (including the source) broadcasts all the packets it has sent to other nodes, or received from other nodes, during rounds 1 and 2 – as with the failure notifications in round 3, agreements on these broadcast packets are achieved using the traditional Byzantine agreement algorithm.

Using the broadcast information extended round 3, all fault-free nodes form identical *diagnosis graphs* after the broadcast phase. The diagnosis graph contains n vertices $0, 1, ..., n - 1$, corresponding to the n nodes in our network; there is an undirected edge between each pair of vertices, with each edge being labeled as g at time 0 (with g denoting "good"). The labels may change to f during extended round 3. Once a label changes to f (denoting "faulty"), it is never changed back to g. Without loss of generality, consider the edge between vertices X and Y in the diagnosis graph. The label for this edge may be set to f in two ways:

- After the broadcast phase, all nodes obtain identical information about what every node "claims" to have sent and received during rounds 1 and 2. Then, for each packet in rounds 1 and 2 (sent by any node), each fault-free peer will compare the claims by nodes X and Y about packets sent and received on links XY and YX. If the two claims mismatch, then the label for edge XY in the diagnosis graph is set to f.

- If node X is a peer and claims to have detected a misbehavior in round 3, but the packets it claims to have received in rounds 1 and 2 are inconsistent with this claim, then edge XY in diagnosis graph is set to f for all Y≠X. In this case, all edges associated with X are set to f.

Similar to the argument in [22], an edge will be marked f only if at least one of the nodes associated with this edge is faulty. Moreover, the label of at least one edge will be changed from g to f after the broadcast phase.

5.2   Operations after failure detected

After a failure is detected, and the extended round 3 is finished, a new generation of $(n - t)c$ bits of new data begins. Let us call nodes i and j accuse(trust) each other if edge ij is marked $f(g)$ in the diagnosis graph. Notice that if a node is accused by more than t other nodes, this node must be faulty.

If the source node 0 is accused by more than t peers, node 0 must be faulty, and the fault-free peers can terminate the algorithm and all agree on some default value. Now consider the case when the source node is accused by no more than t peer. If a peer is identified as faulty, it is *isolated* from the network by marking all edges adjacent to it as f. Then the algorithm operates as follows.

Round 1: Without loss of generality, assume that node 0 is accused by $m \leq t < n/3$ peers. If node 0 is indeed fault-free, it encodes the $n - t$ packets of data into $2(n - 1 - m) > n - t$ packets[2], each of which is obtained as a linear combination of the $n - t$ packets of data. And similar to the case when no failure is detected, every subset of $n - t$ encoded packets consists linear independent combinations of the $n - t$ data packets. For convenience, we will index the two packets node 0 sends to node i as y_i and y_{n-1+i} as before. For a node i that accuses node 0, y_i and y_{n-1+i} do not exist (or equal to NULL).

Round 2: Every fault-free peer i that is trusted by node 0 sends y_i to every peer j that it trusts.

Round 3: For every fault-free peer i that is accused by node 0, if it receives at least $n - t$ packets from the peers it trusts in round 2, peer i first checks these packets for consistency. If these packets are inconsistent, node i detects an attack. Otherwise node i generates one packet z_i as a linear combination of the packets it receives from the trusted peers such that z_i and any $n - t - 1$ packets from the trusted peers consists linear independent combinations of the $n - t$ data packets. It is possible that peer i receives fewer than $n - t$ packets in round 2. For example: t peers including node i are accused by node 0, and t other peers are accused by node i, then there are only $n - 2t - 1 < n - t$ peers not accused by either node 0 or i. In this case, some of the trusted peers of node i sends the second packet (y_{n-1+j}) to node i, such that node i receives $n - t$ packets in total. Then node i generates the packet z_i as a linear combination of the $n - t$ packets it has received similar to the previous case. After z_i is generated, peer i sends it to all the peers it trusts.

For the correctness of the algorithm, any subset of $n - t$ of the union of the y and z packets must be linearly independent. As we know from the design of Reed-Solomon codes and linear network coding, if c is chosen large enough, this linear independence requirement can be satisfied. In principle, since every peer i accused by the source receives at least $n - t$ linear independent packets, it can first solve $\tilde{x}$ and then generate z_i in the same way the y packets are generated.

After z_i is generated, every node i sends z_i to all the peers it trusts. Then every fault-free peer finds the solution for each subset of $n - t$ packets from among the packets received from the other nodes in rounds 1, 2 and 3, and detects an attack if no unique solution is found. Then a 1-bit notification is broadcast as before. Similar to Theorem 1, we have the following

Theorem 2 *After a failure or attack is detected, further misbehavior by a faulty node will either be detected by at least one fault-free peer, or all the fault-free peers will reach agreement correctly.*

Proof: Here we also consider the cases when source node 0 is fault-free and faulty separately.

Source is faulty: Denote P and Q as the set of fault-free peers that are trusted by the source and accused by the source, respectively. Since source is faulty, there are at least $n - t$ fault-free peers, i.e. $|P| + |Q| \geq n - t$. As stated before, an edge is marked f only if at least one node associated with this edge is faulty. This implies that an edge associated with two fault-free nodes is always marked g, and two fault-free nodes will never accuse each other. Thus, every peer $i \in P$ sends y_i to all peers in $P \bigcup Q$ during round 2. Similarly, every peer $i \in Q$ sends z_i to all peers in $P \bigcup Q$ during round 3. As a result,

---

[2]Since $m \leq t$, $2(n - 1 - m) \geq 2(n - 1 - t) = n - t + (n - t - 2)$. From $t \geq 1$ and $n \geq 3t + 1$, we have $n - t - 2 > 0$, then $2(n - 1 - m) > n - t$.

the fault-free peers $P \bigcup Q$ share at least $|P| + |Q| \geq n - t$ identical packets. Thus, the fault-free peers will not agree on different data symbols.

Source is fault-free: Remember that two fault-free nodes will never accuse each other. Thus all fault-free peers receives two correct packets from source node 0 in round 1 and they exchange 1 packet with each other in round 2. Similar to the proof of Theorem 1, the fault-free peers will not agree on any vector other than $\tilde{x}$. $\qquad\qquad\square$

Extended Round 3: If a failure is detected by a node associated with no more than t f-links (otherwise the node must be faulty and will be isolated as before), the extended round 3 is entered and carried out as before. By the end of every extended round 3, at least one more edge in the diagnosis graph will be marked f.

5.3 Complexity of the Algorithm

When no failure detected: In every generation, excluding the extended round 3:

- Round 1: source node 0 sends $2(n-1)$ packets to the peers, which is $2(n-1)c$ bits;

- Round 2: every peer sends $(n-2)c$ bits, thus $(n-1)(n-2)c$ bits in total;

- Round 3: each peer broadcasts 1 bit notification. Let us denote B as the bit-complexity of achieving agreement on 1 bit. So in round 3, totally $(n-1)B$ bits are transmitted;

So if no failure is detected in round 3, totally

$$2(n-1)c + (n-1)(n-2)c + (n-1)B = n(n-1)c + (n-1)B \qquad (3)$$

bits are transmitted in the whole generation.

After failure detected:

- Source node 0: sends at most $2(n-1)$ packets in round 1, which is no more than the usage when no failure detected;

- Among nodes trusted by node 0: they send 1 packet to each other, which is the same as no failure detected;

- Nodes accused by node 0: each such node i receives and sends 1 packet from every peer it trusts, which is no more than the usage when no failure detected; if node i receives fewer than $n-t$ packets in round 2, it receives $n-t$ in rounds 2 and 3 together, which is less than the usage of $n-2$ when no failure is yet detected. Every such node i sends at most $n-2$ packets to the peers it trusts in round 3, which is no more than the usage when no failure is yet detected.

- Only nodes accused by no more than t nodes need to achieve agreement on the 1-bit notifications.

Now it should not be hard to see that after failure is detected, the usage of capacity is at most as much as the case when no failure is yet detected, excluding extended round 3. So in the normal operation (rounds 1 to 3), at most $n(n-1)c + (n-1)B$ bits are transmitted in every generation.

Lit.	Complexity	Authentication required?	Probabilistic/deterministic agreement?
[2]	$\Theta(n^2 l)$	**No**	**Deterministic**
[28]	$\Omega(n^2 l + n^6 \kappa)$	Yes	Probabilistic
[9]	$\Omega(n^2 l + n^3 \kappa)$	Yes	Probabilistic
[11]	$(3n-1)l + \Theta(n^3(n+\kappa))$	**No**	Probabilistic
Ours	$< \frac{3}{2}nl + l^{1/2}\Theta(n^4)$	**No**	**Deterministic**

Table 1: Complexity of existing BA algorithms. The complexity of algorithms from[2, 28] is cited from [11].

Extended round 3: Source broadcasts at most $2(n-1)$ packets, and every peer broadcasts at most n packets, when a failure is detected. So in every extended round 3, no more than $2(n-1)+n(n-1) = (n+2)(n-1)$ packets are broadcast, which results in $(n+2)(n-1)cB$ bits being transmitted.

Now we look at the total number of bits being used to agree on l bits of data. Notice that $(n-t)c$ bits are being agreed on in every generation, so there are $l/(n-t)c$ generations in total. Thus in rounds 1 to 3 in all generations, no more than totally $\frac{n(n-1)}{n-t}l + \frac{(n-1)B}{(n-t)}\frac{l}{c}$ bits are transmitted. There are at most $(t+1)t$ extended round 3 before all faulty nodes are identified. So the total usage of extended round 3 in all generations are at most $(t+1)t(n+2)(n-1)cB$ bits. Now we have a upper bound on the total number of bits being transmitted to achieve agreement on l bits as:

$$\frac{n(n-1)}{n-t}l + \frac{(n-1)B}{n-t}\frac{l}{c} + (t+1)t(n+2)(n-1)cB \tag{4}$$

$$= \frac{n(n-1)}{n-t}l + 2Bl^{1/2}\sqrt{\frac{(t+1)t(n+2)(n-1)^2}{n-t}} \quad \left(\text{by a suitable choice of } c = \sqrt{\frac{(n-1)l}{(n-t)(t+1)t(n+2)(n-1)}}\right) \tag{5}$$

$$< \frac{n(n-1)}{n-t}l + 2Bl^{1/2}\sqrt{\frac{(n+2)^2(n-1)^3}{6n}} \quad \text{(maximized when } t = (n-1)/3\text{)} \tag{6}$$

$$= \frac{n(n-1)}{n-t}l + Bl^{1/2}\Theta(n^2). \tag{7}$$

Notice that broadcast algorithm of complexity $\Theta(n^2)$ are known [2], so we assume $B = \Theta(n^2)$. Then we have the complexity of our algorithm for all $t < n/3$ upper bounded by

$$\frac{n(n-1)}{n-t}l + l^{1/2}\Theta(n^4) < \frac{3}{2}nl + l^{1/2}\Theta(n^4). \tag{8}$$

For a given network with size n, if we normalize the left hand side of Eq.8 over l, we obtain a upper bound on the sum capacity of the system for achieving agreement throughput of R bits/unit time as

$$\frac{n(n-1)}{n-t}R + \frac{\Theta(n^4)}{l^{1/2}}R \to \frac{n(n-1)}{n-t}R, \quad as \ l \to \infty \tag{9}$$

Table 1 states the overall bit complexities for agreeing on an l-bit message, both for the most efficient protocols in the literature and for the protocol presented in this paper. In particular, compared with the best known algorithm in [11], our algorithm has strictly lower complexity when $l \geq \omega n^6$ for some constant $\omega > 0$. Moreover, the low complexity in [11] is achieved by allowing a positive probability of error (fault-free nodes may decide on different values), while our algorithm is guaranteed to achieve agreement deterministically such that all fault-free nodes always agree on the same (correct) value.

6 Optimality of our Algorithm

Authors of [11] have shown that their algorithm is *order-optimal* since any algorithm that achieves Byzantine agreement requires the communication of $\Omega(nl)$ bits, and their algorithm requires $(3n-1)l + \Theta(n^3(n+\kappa))$ bits. This results in requiring at least $(3n-1)R$ bits/unit time of sum capacity in order to achieve agreement throughput of R. If we define the average cost per agreed bit α as the ratio of the sum capacity over the achievable throughput of agreement, then $\alpha_{[11]} = 3n-1$. On the other hand, according to Eq.9, our algorithm requires a lower ratio $\alpha_{our} = \frac{n(n-1)}{n-t} \leq 3n/2 < 3n-1$. In fact, we believe that our algorithm is not only order-optimal, but also optimal in the sense of minimizing α.

Conjecture 1 *In order to achieve agreement on l bits, at least $\frac{n(n-1)}{(n-t)}l$ bits need to be transmitted in the network. This means that to achieve agreement throughput of R bits/unit time, the sum capacity of the system must be at least $\frac{n(n-1)}{(n-t)}R$ bits/unit time.*

Argument: This conjecture is argued based on another conjecture as follows:

Conjecture 2 *Given k nodes each of which is assigned an arbitrary initial value of l bits, at least $(k-1)l$ bits are necessary such that if the k initial values are not identical, as least one node will detect it.*

Conjecture 2 is easily shown to be true when $l = 1$. Since we assume 1 bit as the smallest unit of data, if two nodes communicate with each other, at least 1 bit is transmitted on one of the two links connecting the two nodes. We call such a pair of nodes *connected*. It should not be hard to see that just to check if all k initial bits are identical, the network must be connected. And it is well-known from graph theory, that to connect k nodes, at least $k-1$ links are needed. Thus, at least $k-1$ bits are necessary just to check initial values of 1 bit among k nodes. Intuitively, every bit of the l-bit initial value is independent. So the checking result of a particular bit is independent of the results of the other bits. This implies that every bit needs to be checked individually. For each bit, we need to form a connected graph using at least $k-1$ bits. Thus, to check l bits, $(k-1)l$ bits are needed in total. Although this intuition sounds valid, the formal proof is much more intriguing and we are still working on it. This is the reason why we state this as a conjecture rather than a theorem.

Based on Conjecture 2, we argue that in a network with n nodes and up to t failures, when no failure is yet detected initially, at least $(n-t-1)l$ bits need to be allocated for links among every subset of $n-t$ nodes. To argue this, we need the following reduction:

Remember that our goal is to achieve agreement with up to t faults (any subset of $\leq t$ nodes). Consider one subset $F = \{f_0, \ldots, f_{t-1}\}$ of t nodes containing the source node 0 (f_0) that may be faulty. The other $n-t$ nodes are known to be fault-free and denote the set of these nodes as $G = \{g_1, \ldots, g_{n-t}\}$.

Given any algorithm that achieve agreement on l bits, we construct the state machine illustrated in Fig.1. In this state machine, for node g_i, $F_i = \{f_{i,0}, \ldots, f_{i,t-1}\}$ is the set of virtual nodes corresponding to F and run the same *correct* code as nodes in F should run, and $g_{i,j}$ is the virtual node corresponding to g_j and runs the same code as node g_j. The good node g_i sends identical messages to node g_j and $g_{i,j}$. Each virtual source node $f_{i,0}$ is given an initial value v_i of l bits. In the real network, node f_j behaves to node g_i as node $f_{i,j}$ in the state machine. It should be easy to see that if $v_1 = v_2 = \cdots = v_{n-t}$ the nodes in B are actually not misbehaving.

Let us assume that all the nodes in G know that nodes in F behave in the way described above. Observe that the behavior f_i is fully determined by v_i and the messages node g_i sends to f_i. So if g_i knows the value of v_i, it can emulate the behavior of f_i. Then we can reduce the agreement problem of n nodes to the following problem of $n-t$ fault-free nodes of G, in which each node g_i is given an initial
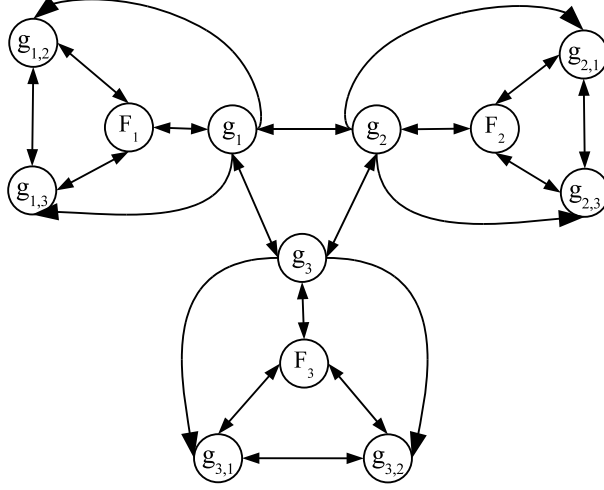
Figure 1: State machine for $|G| = 3$.

value v_i. Since algorithm can achieve agreement in the original network, agreement must be able to achieved by having node g_i emulating the behavior of f_i.

The above agreement algorithm for the $n - t$ fault-free nodes can be used to check if the $n - t$ initial values are identical: Denote v as the value agreed upon by the agreement algorithm. If $v_1 = v_2 = \cdots = v_{n-t}$, it is as if no node is misbehaving in the original network and v must equal to v_i for all i. On the other hand, $v \neq v_i$ for some i, then the $n - t$ initial values must not be all identical, and node g_i will detect it. By Conjecture 2, it is impossible to check the if all initial values are identical with fewer than $(n - t - 1)l$ bits being transmitted. This means that the BA algorithm must allocate at least $(n - t - 1)l$ bits on links between nodes in G.

Since the location of the faulty peers are unknown a priori, so at least $(n - t - 1)l$ bits must be assigned for every subset of $n - t$ peers. For the subsets containing the source and $n - t - 1$ peers, a similar reduction is applied and leads to the same result.

Now we have shown that $(n - t - 1)l$ bits are necessary for every subset of $n - t$ nodes in the network. If we sum over all $\binom{n}{n-t}$ subsets of $n - t$ nodes, the summation is $\binom{n}{n-t}(n - t - 1)l$ bits, while each links is counted $\binom{n-2}{n-t-2}$ times. Then we can computer the following lower bound on the total bit complexity for any BA algorithm on l bits

$$\frac{\binom{n}{n-t}(n - t - 1)l}{\binom{n-2}{n-t-2}} = \frac{\frac{n!}{(n-t)!t!}(n - t - 1)l}{\frac{(n-2)!}{t!(n-t-2)!}} = \frac{n(n - 1)}{n - t}l \tag{10}$$

This completes the argument of Conjecture 1. □

If Conjecture 2 is proven to be true, Conjecture 1 is true too. Then for a network of size n with up to t faults and total capacity C bits/unit time, by Conjecture 1, the agreement capacity of this network cannot exceed $\frac{n-t}{n(n-1)}C$ bits/unit time. On the other hand, our algorithm can achieve agreement throughput arbitrarily close to this upper bound. Thus the agreement capacity of such a network is $\frac{n-t}{n(n-1)}C$ bits/unit time.

7   Conclusion

In this work, we have proposed a highly efficient Byzantine agreement algorithm on values of length $l > 1$ bits. This algorithm uses error detecting network codes to ensure that fault-free nodes will never

disagree, and routing scheme that is adaptive to the result of error detection. Our algorithm has a bit complexity of $\frac{n(n-1)}{n-t}l$, which leads to a linear cost ($O(n)$) per bit agreed upon, and overcomes the quadratic lower bound ($\Omega(n^2)$) in [8]. Such linear per bit complexity has only been achieved in the literature by allowing a positive probability of error. Our algorithm achieves the linear per bit complexity while guaranteeing agreement is achieved correctly even in the worst case. We also conjecture that our algorithm can be used to achieve agreement throughput arbitrarily close to the agreement capacity of a network, when the sum capacity is given.

References

[1] H. Attiya and J. Welch. *Distributed Computing*. McGraw-Hill, 1998.

[2] P. Berman, J. A. Garay, and K. J. Perry. Bit optimal distributed consensus. *Computer science: research and applications*, pages 313–321, 1992.

[3] N. Cai and R. W. Yeung. Network error correction, part ii: Lower bounds. *Communications in Information and Systems*, 6(1):37–54, 2006.

[4] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 2002.

[5] T. D. Chandra, R. Griesemer, and J. Redstone. Paxos made live: an engineering perspective. In *PODC '07*, pages 398–407, New York, NY, USA, 2007. ACM.

[6] B. A. Coan and J. L. Welch. Modular construction of a byzantine agreement protocol with optimal message bit complexity. *Inf. Comput.*, 97(1):61–85, 1992.

[7] E. C. Cooper. Replicated distributed programs. In *SOSP'85*, 1985.

[8] D. Dolev and R. Reischuk. Bounds on information exchange for byzantine agreement. *J. ACM*, 32(1):191–204, 1985.

[9] D. Dolev and H. R. Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[10] C. Dwork and Y. Moses. Knowledge and common knowledge in a byzantine environment: crash failures. *Inf. Comput.*, 88(2):156–186, 1990.

[11] M. Fitzi and M. Hirt. Optimally efficient multi-valued byzantine agreement. In *PODC '06: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 163–168, New York, NY, USA, 2006. ACM.

[12] C. Fragouli, D. Lun, M. Medard, and P. Pakzad. On feedback for network coding. In *Proc. of 2007 Conference on Information Sciences and Systems (CISS)*, 2007.

[13] R. Friedman, A. Mostefaoui, S. Rajsbaum, and M. Raynal. Distributed agreement and its relation with error-correcting codes. In *Proc. 16th Int. Conf. Distributed Computing*, 2002.

[14] T. Ho, B. Leong, R. Koetter, M. Medard, M. Effros, and D. Karger. Byzantine modification detection in multicast networks using randomized network coding, 2004.

[15] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, and M. Medard. Resilient network coding in the presence of byzantine adversaries. In *INFOCOM'07*, 2007.

[16] S. Kim, T. Ho, M. Effros, and S. Avestimehr. New results on network error correction: capacities and upper bounds. In *ITA'10*, 2010.

[17] R. Koetter and M. Medard. An algebraic approach to network coding. In *Information Theory, 2001. Proceedings. 2001 IEEE International Symposium on*, pages 104–, 2001.

[18] M. N. Krohn. On-the-fly verification of rateless erasure codes for efficient content distribution. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 226–240, 2004.

[19] L. Lamport and K. Marzullo. The part-time parliament. *ACM Transactions on Computer Systems*, 16:133–169, 1998.

[20] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4:382–401, 1982.

[21] S.-Y. Li, R. Yeung, and N. Cai. Linear network coding. *Information Theory, IEEE Transactions on*, 49(2):371–381, Feb. 2003.

[22] G. Liang and N. Vaidya. Capacity of byzantine agreement: Complete characterization of the four node network. *Technical Report, CSL, UIUC*, April 2010.

[23] G. Liang and N. Vaidya. Capacity of byzantine agreement: Tight bound for the four node network. *Technical Report, CSL, UIUC*, February 2010.

[24] N. A. Lynch. *Distributed algorithms*. Morgan Kaufmann Publishers, 1995.

[25] T. Mizrahi and Y. Moses. Continuous consensus via common knowledge. In *TARK'05*, 2005.

[26] T. Mizrahi and Y. Moses. Continuous consensus with failures and recoveries. In *DISC '08*, 2008.

[27] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *JOURNAL OF THE ACM*, 27:228–234, 1980.

[28] B. Pfitzmann and M. Waidner. Information-theoretic pseudosignatures and byzantine agreement for $t \geq n/3$. *Technical Report, IBM Research*, 1996.

[29] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM*, 36(2):335–348, 1989.

[30] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[31] N. Vaidya and G. Liang. Capacity of byzantine agreement (preliminary draft - work in progress). *Technical Report, CSL, UIUC*, January 2010.

[32] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan. An efficient signature-based scheme for securing network coding against pollution attacks. *INFOCOM 2008*, pages 1409–1417, April 2008.